P.I. Name: Douglas R. Smith
Insitution: Kestrel Institute
Telephone: (415) 493-6871
E-mail: smith@kestrel.edu
Contract Title: Theory of Algorithm Structure and Design
Contract Number: N00014-90-J-1733
Reporting Period: 1 Oct 89 - 30 Sep 92

## 1. Productivity Measures

Refereed papers submitted but not published: 4

Refereed papers published: 13

Unrefereed reports and articles: 0

Books or parts thereof submitted but not yet published: 2

Books or parts thereof published: 4

Patents filed but not yet granted: 0

Patents granted: 0

Invited presentations: 29

Contributed presentations: 12

Honors received: 8







1. Dr. Smith elected member of IFIP Working Group 2.1 "Algorithmic Languages and Calculi".
2. Dr. Lowry served as program committee member and session chair for workshop "Change of Representation and Problem Reformulation", Menlo Park, March 1990.
3. Dr. Smith was appointed Lecturer in Computer Science, Spring Quarter 1990, Computer Science Department, Stanford University.
4. Dr. Smith was Co-Chairman, local arrangements chair, program committee member, and session chair, IFIP TC2 Working Conference on Constructing Programs from Specifications, Pacific Grove, California, May 1991.
5. Dr. Smith was a Program Committee member, 1991 IEEE Conference on Tools for Artificial Intelligence, June 1991.
6. Dr. Smith was appointed Lecturer in Computer Science, Spring Quarter 1991, Computer Science Department, Stanford University.
7. Dr. Smith was Panelist, AI and Software Engineering Panel, 1991 IEEE Conference on Tools for Artificial Intelligence, San Jose, California, 13 November 1991.
8. Dr. Smith was a Program Committee member, Session chair, and Panelist, Seventh Knowledge-Based Software Engineering Conference, McLean, VA, 21-23 Sep 1992.

9. Dr. Smith was appointed Lecturer in Computer Science, Spring Quarter 1992, Computer Science Department, Stanford University.

Prizes or awards received: 0

Promotions obtained: 0

Graduate students supported: 2

Post-docs supported: 1

Minorities supported: 0

P.I. Name: Douglas R. Smith
Insitution: Kestrel Institute
Telephone: (415) 493-6871
E-mail: smith@kestrel.edu
Contract Title: Theory of Algorithm Structure and Design
Contract Number: N00014-90-J-1733
Reporting Period: 1 Oct 89 - 30 Sep 92

## 2. Summary of Technical Results

Algorithms and data structures are among the primary constituents of computer software and thus are among basic objects of study in Computer Science. This project is concerned with the structure and automated design of algorithms and data structures. Our scientific hypothesis is that there exist general algorithm, data structure, and design concepts that underlie and explain most of the detailed structure of conventional software systems. By abstracting and formalizing these concepts and showing how to mechanize their application, we can prepare the way for the coming generation of automated software design environments.

Our approach involves identifying classes of algorithms that solve a broad range of useful problems. In particular we have emphasized formalizing abstract algorithms that make minimal assumptions about the structure of a problem. Once a class of algorithms has been identified we represent its essence as a theory, called an *algorithm theory* [11]. Under ONR support we have developed algorithm theories and design tactics for divide-and-conquer [2], simple problem reduction [2], global search (binary search, backtrack, branch-and-bound) [4], problem reduction generators (dynamic programming, generalized branch-and-bound, game tree search) [6], local search [1], and others. These have all been at least partially implemented and tested in the KIDS system [10]. KIDS has been used to derive over 60 algorithms.

More recent work has focused on theories and operations on theories as the formal underpinings of algorithm design as well as data structure design and refinement and general software development. Algorithm design is based on constructing a theory morphism[1] from an algorithm theory into a given application domain theory. Datatype design and and refinement are also based on constructing a theory morphism from an one datatype theory into another. Generally, specifications are theories and the implementation of specifications is based on constructing a theory morphism into a (relatively) concrete, computationally-oriented theory. This formal view of software development has motivated research into the kinds of theories that are useful for specifying and reasoning about application domains and systems, as well as capturing knowledge about algorithms, data structures, and other kinds of programming knowledge. It has also led us to focus our attention on formal/automatable techniques for constructing theory morphisms.

Project results during the last year are listed below.

1. **Problem Reduction Generators**

   Problem reduction is a pervasive technique for solving problems by reducing a problem instance to a structure of subproblem instances. Solutions to the subproblem instances are

[1] A *theory morphism* from theory $A$ to theory $B$ is a translation of the language of $A$ to the language of $B$ such that theorems of $A$ translate to theorems of $B$.

composed to form a solution to the initial instance. Problem reduction generators are used to enumerate all solutions to a given problem. Problems are given by means of a formal specification (we use a specification language that extends first-order predicate calculus notation with set-theoretic datatypes).

We produced an algorithm theory and design tactic for the class of problem reduction generators [11]. This algorithm theory, called *complete problem reduction theory*, provides the logical basis for dynamic programming, branch-and-bound, game-tree search, and other well-known algorithm paradigms.

The design tactic extends the structure of a specified problem with the structure of complete problem reduction theory. If we decide to compute the characteristic recurrence equation of problem reduction theory from the bottom up, then we get the usual dynamic programming algorithms. Top-down control leads to branch-and-bound algorithms such as AO* and game-tree algorithms such as alpha-beta and SSS*.

The design tactic constructs a complete problem reduction theory for the specified problem through a combination of selecting of standard information from a library, deductive propagation of the consequences of choices, and verification of axioms. A major result of our project has been the development of deductive techniques for using axioms to help construct complex objects rather than simply to use them as verification conditions. For example, in complete problem reduction theory we use an axiom ("strong soundness") to deduce a specification for the reduction step of the algorithm given a choice of a standard composition step (which is usually a standard constructor algebra for the output domain).

The design tactic has been applied to about a dozen problems in order to explore its generality and the feasibility of its steps. The paper [11] presents a detailed treatment of the problem of enumerating optimal binary search trees. In 1990 we implemented in KIDS a new collection of tools based on theories, signature morphisms, theory morphisms, and theory translation. The tactics for divide-and-conquer and problem reduction generators were implemented using these tools and the result has been far more general and flexible tactics than previously. Partly this generality is due to the theories being indexed by a signature, as in algebraic theories. In previously published algorithm theories and design tactics we have had to fix the signature thereby limiting the applicability of the theory unnecessarily.

## 2. KIDS and 1990 Challenge Problems

KIDS (Kestrel Interactive Development System) is the testbed for our research on semiautomated program design [10, 5]. The system has components for performing algorithm design, deductive inference, program simplification, partial evaluation, finite differencing optimizations, data type refinement, compilation, and other development operations. To use KIDS for a new problem, the user first builds up a domain theory that formalizes the concepts of the problem and provides laws for reasoning about the concepts. The domain theory provides the vocabulary for expressing a formal specification of the problem. Next the user applies the design, optimization, and refinement tools as needed in order to obtain an efficient and correct source-level program that implements the specification.

In 1990 we used KIDS to respond to several challenge problems presented to us by non-Kestrel personnel. By accepting these challenges our aims were to deepen our understanding of automated algorithm design through applying KIDS to concrete problems and to demonstrate the effectiveness of KIDS by working problems that were not selected by personnel with an intimate knowledge of the system.

We briefly discuss two challenge problems: enumerating cyclic projective planes of order $n$ and enumerating Costas arrays. The first problem was quite difficult, but resulted in an

algorithm that was significantly better than most programmers (including ourselves) could have designed given our simple understanding of the problem. Unfortunately we found out latter that there is a considerable and deep literature on the problem under the rubric of combinatorial design theory. Exploiting this knowledge would have resulted in much better algorithm.

The Costas array derivation was more succcessful [5]. This is a problem arising in the design of sonar and radar signals with optimal ambiguity functions. The domain theory was built up over a period of about a week and enabled the derivation of an efficient backtrack algorithm. After we hand-refined the abstract data structures of the resulting program and manually translated it into C, we were able to enumerate all order 17 Costas arrays. This is as high as has been published in the literature by the various groups of mathematicians exploring this problem. This result is evidence that the machine-generated algorithm is comparable to the programs created by insightful and experienced human programmers.

There were several important lessons from these derivations and others like them. First, much of the hard work in performing a new derivation lies in building up the domain theory. As of mid 1990 we had built up a library of some 25 theories, including basic theories such as finite set theory and finite map theory, as well as more specialized theories for scheduling and Costas arrays. The more basic theories are highly reusable so over time we can expect that theory building will involve more selection and adaptation. Our experience points to theory-building as an irreducible and time-consuming activity in future automated design environments.

In 1991 we ran the derived Costas code in parallel on the 13 Sun Sparcstations at Kestrel in background mode. We were able to enumerate all Costas arrays of size 18, 19, and 20. The latter took almost 4 months.

Another lesson concerns the nature of the domain theories. What constitutes a well-formulated theory? Our experience has been that a well-formulated domain theory has simple laws for reasoning about the basic concepts. A related observation is that most of the laws needed to support design, optimization, and refinement in KIDS are distributive laws. Together these observations suggest that when building a new domain theory, we should seek conceptualizations that admit simple distributive laws. Our initial formulation of Costas array theory had quite complex laws and we eventually discarded it. A second attempt was successful and the basic concepts had very elegant distributive laws and the ensuing design process was straightforward.

## 3. Theory Development in KIDS

In light of the fundamental importance of domain theories in KIDS (and for formal methods of software design in general), in 1990 we began to explore tools for supporting the development of domain theories. We prototyped a simple grammar for theories comprising imports, type definitions, concept definitions (functions), laws, and inference rules. We then built an interactive graphical interface and a collection of operations that could be used to help users build theories. Loading a theory recursively loads all imports and installs the concept definitions, laws, and inference rules in the KIDS active knowledge base. There is also a tool for automatically deriving distributive laws from a concept definition. The resulting laws can be transformed into nonlogical inference rules. Another operation allows the abstraction of new concepts from the body of any expression. For example, if a derived distributive law contains a complex subexpression $S$, then $S$ can be abstracted out, named, and added to the theory. In this way the user may enter some definitions via a text editor, then use the system to derive new laws, rules, and concepts.

In mid-1992 there are roughly 2000 rules in the KIDS system encapsulated in about 100 theories. Since theory development is the ultimate problem of software engineering, we see great need and many exciting future possibilities for developing the theory development system in KIDS.

4. Specifications, Specification Morphisms, and Operations on Theories

   Theories and operations on theories have emerged at Kestrel as a pervasive foundation in our software design automation studies. Our algorithm design work is expressed in terms of abstract algorithm theories and interpretations between abstract and concrete algorithm theories. Related concepts of problem theories and program theories come into play. The use of (equational) theories to formalize abstract data types is well-known. We have been formalizing a grammar and compilation mechanism for specification morphisms as a way to formally express and use datatype refinements. For example, the implementation of sets as bit vectors is essentially an interpretation between set theory and bit vector theory. Generally, interpretations between theories provide the basis for refinement of specifications. The use of theories to express domain theories and various operations to help build them was discussed above. Thus theories and their operations play a fundamental role in expressing program design knowledge. Theories can also be used as a module mechanism in a specification/programming language, providing modules with a formally defined interface (not only the imported and exported types and operations, but the axioms that constrain their meaning too). For these reasons and others we have been exploring theories and their operations, extending the results of the OBJ3, LARCH, and Extended ML projects.

5. Problem Reformulation Our work on problem reformulation has been extended along several dimensions. Previous work on abstraction of domain and problem specifications has been extended to domains with hidden states, thereby facilitating scaling up to module and system abstraction. The foundation for abstraction through symmetry has progressed; we anticipate application to continuous domains during the coming year. A variant of Karmarkar's algorithm has been previously developed by hand using abstraction through symmetry, we anticipate that further work along these lines will provide the formal foundation for automating this development.

6. **Local Search** Using permutation group theory, we have further developed our research on local search algorithms by formalizing a class of combinatorial local search algorithms based on permuting components of data structures representing feasible solutions. Previously, a design tactic based on these ideas had been developed and demonstrated on the synthesis of the Simplex algorithm using KIDS. We are currently extending our research on local search through a design tactic for primal/dual algorithms.

7. **KIDS and 1991 Challenge Problems**

   In 1991 we applied KIDS to derive a somewhat larger problem than before [9]. We obtained a structured English specification of the Track Assignment portion of an Air Traffic Control system that was developed by Hughes Aircraft. This was about 6 pages of text. A domain theory was developed in KIDS (about 24 pages of formal text) in which most of the laws and rules were automatically derived using the theory development mode in KIDS. Then various design, optimization, and refinement operations were applied to generate code from a specification of a track assignment module. This code was similar in structure to hand generated ADA produced by Hughes personnel. The main difference was that our code was expressed in a functional subset of the Regroup language (our local extension of the Refine langauge).

8. **Derivation of Parallel Algorithms** Although our algorithm design tactics have been used to produce algorithms with good parallel properties [3, 5], parallel algorithms has not been an explicit focus of our research. However for the purposes of the Workshop on Parallel Algorithm Derivation and Program Transformation held in New York City in 1991, I explored the application of the algorithm design tactics to some classic parallel sorting algorithms [7]. I choose Batcher's parallel mergesort algorithm which is quite difficult to explain and has some desirable features – it was the basis of one of the first sorting networks. The reason that the algorithm is so difficult to explain is that it is based on a shuffle constructor for sequences rather than the more familiar cons or concat: *shuffle*$([1,3,5],[2,4,6]) = [1,2,3,4,5,6]$, *cons*$(1,[2,3]) = [1,2,3]$, *concat*$([1,2,3],[4,5,6]) = [1,2,3,4,5,6]$. Since the constructors of a type have a heavy influence on the formulation of the operations and relations on a type, one would expect that a shuffle-based sequence theory would be quite different than cons-based theories. Indeed the hardest part of this derivation was to build up the shuffle-based theory of sequences – questions such as 'under what conditions is the shuffle of two sequences sorted?' lead to whole new concepts and the laws for reasoning about them. Once this theory is built up the derivation of a $O(\log n)$-time merge operation is straightforward and yields a $O(\log^2 n)$-time mergesort. Also derivable in this new shuffle-based theory is the well-known odd-even transposition sort and a variety of other shuffle-related algorithms.

9. **Constructing Theory Morphisms** In 1991 we identified two new general techniques for constructing theory morphisms: unskolemization and connections between theories. Roughly put, *unskolemization* works in the following way. A theory morphism from theory $A$ to theory $B$ is based on a signature morphism which is a map from the symbols of $A$ to the symbols of $B$. A theory morphism is signature morphism in which the axioms of $A$ translate to theorems of $B$. Suppose that during a design process we have somehow managed to construct a partial signature morphism – only some of the symbols of $A$ have a translation as symbols of $B$. The question is how to derive a translation of the remaining symbols of $A$. The unskolemization technique uses the axioms of $A$ and deductive inference to solve for appropriate translations of these symbols. As a simple example, suppose that function symbol $f$ is untranslated and that it is the only untranslated symbol in an axiom $\forall(x)G[f(x)]$ of $A$. We unskolemize $f$ by replacing its occurrence(s) with a fresh existentially quantified variable: $\forall(x)\exists(z)G[z]$. This unskolemized axiom can now be translated and we can attempt to prove it in theory $B$. A proof yields a witness for the existential that is a term that depends on $x$. This term can serve as the translation of $f$ knowing that such a translation preserves the theoremhood of the considered axiom. We may need to verify of axioms involving $f$ to assure the appropriateness of the derived translation.

   This technique underlies the problem reduction family of algorithms and tactics in KIDS. For example, in constructing a divide-and-conquer algorithm we need to find translation of decompose, solve, and compose operators. The tactic works by letting the user select a standard decomposition operator from a library (or dually selecting a standard compose operator) and then using unskolemization on a "soundness axiom" that relates decompose and compose. The unskolemized soundness axiom can then be proved in the given problem theory to yield a specification of the compose (resp. decompose) operator. To be more specific, if we are deriving a divide-and-conquer algorithm for the sorting problem, then we want to construct a theory morphism from divide-and-conquer theory into sorting theory. We might choose a standard decompose operator for input sequences, say split-a-sequence-in-half, and the unskolemization technique leads to a derivation of a specification for the usual merge operation as the translation of compose. The result is a mergesort algorithm.

   Sometimes the axioms of a theory are too complex to allow direct application of unskolem-

ization. This situation arises in the theory of global and local search algorithms. We have discovered and developed recently the concept of *connection between theories* which underlies and generalizes our correct but somewhat ad-hoc solution to this problem in the global and local search design tactics. The general result regarding connections between theories is this: Suppose that there is a theory $T$ from which we want to construct a theory morphism into a given application domain theory $B$. If there is a (preexisting) theory morphism from $T$ to a library theory $A$ and we can construct a connection from $A$ to $B$, then we immediately have a theoy morphism from $T$ to $B$. So connections between theories are a way to adapt a library $T$-theory to a new, but related problem.

The concept of connection between theories relies on several ideas. The sorts of a theory are all interpreted as posets (including booleans) and furthermore the set of sorts itself is a poset (under the subsort partial order). A collection of "polarity rules" are used to express (anti-)monotonicity properties of the functions and predicates of a theory. For example, $size(\{x \mid \neg P\})$ is monotonic in $\{x \mid \neg P\}$ but antimonotonic in $P$; so if $Q \implies P$ then $size(\{x \mid \neg P\}) \leq size(\{x \mid \neg Q\})$. These polarity rules are used to analyze the axioms of a theory and then to set up various connection conditions between the corresponding operators of theories $A$ and $B$ – these conditions directly generalize the conditions of a homomorphism. Furthermore the polarity analysis is used to direct conversion maps between corresponding sorts of $A$ and $B$. Given these conditions and conversion maps it can in general be shown that the axioms of $A$ imply the corresponding axioms of $B$, thus establishing the theory morphism.

These two techniques of unskolemization and connection between theories ultimately point toward a next-generation software development environment based on theories and automated operations on theories, automated support for constructing theory morphisms, and large hierarchic libraries of theories. I envision a meta-tactic based on these two techniques and others, that interprets the structure of a given theory (algorithm theory, datatype theory, system specification, etc.) and provides automated support for the construction of a theory morphism that implements it. With such a system, the users' attention shifts to building up the library of programming theories and application domain theories.

I developed the theory of these techniques, proved some fundamental theorems about them, and submitted the results to a journal [8].

## 10. Data Structure Design

A subtype is often defined by a characteristic predicate over elements of a supertype. All operations on the subtype preserve this predicate as an invariant. Most interesting data structures are subtypes in this sense. For example, heaps are a subtype of binary trees that are characterized by the property that the children of a node are larger than the parent. From a design perspective, this suggests two key problems. First, how do the characteristic predicates arise during the development process. Must they simply be invented or are there formal techniques for deriving appropriate predicates? Second, given such a characteristic predicate, how can we formally develop a self-contained theory for the subtype – one without reference to operations of the parent type?

In this past year we have begun developing answers to these two questions. The first, how to derive the characteristic predicate of a subtype, has been explored via a detailed derivation of the classic heapsort algorithm. Heapsort is a specialized selection sort algorithm involving the use of the heap data structure and was the first known $O(n \log n)$ worst-case and average-case algorithm for sorting. Applying divide-and-conquer to the sorting problem (and choosing a simple compose) we derive a specification for the problem of selecting and extractingthe least element of a bag. This gives us an extended bag theory – the usual operations of equality,

membership, construction, etc. augmented with min and extractmin. When we attempt to construct a theory morphism from this extended bag theory into binary tree theory, then unskolemization and various heuristically-guided choices result in the derivation of the heap property. The key choice is that the min operation (in bag theory) be translated as the root operation (in binary tree theory). This results in the heap property. There is a great deal more to explore in the theory of designing data structures, but it appears that the concepts and techniques that we have developed for designing algorithms provides many of the basic techniques.

The other question of how to develop a self-contained subtype theory has been adddressed in the following way. We developed a procedure that derives constructors for a subtype given constructors for the parent type and a characteristic predicate of the subtype. The procedure is sound in that if it terminates it produces a set of constructors that inductively generate the subset of elements of the parent type that satisfy the characteristic predicate. However the procedure need not terminate. In such cases the procedure seems to generate constructors for each value of the subtype – it does not find a recursive pattern that covers the type. For the other operators of the subtype theory we must design them so that they leave the subtype predicate invariant – often this is an algorithm design problem.

## 11. Classification Approach to Algorithm Design

We developed the theoretical foundations needed to support a classification approach to problem-solving: a declarative statement of a problem (e.g a transportation scheduling problem) is classified with respect to a library of problem classes. Each problem class has one or more problem-solving methods associated with it. Classification exposes the implicit structure of the problem that can be exploited by a problem-solver. Thus a problem-solving method that applies to a given problem is obtained as a by-product of the classification process.

Problem-solving knowledge is represented as formal theories and arranged in a refinement hierarchy. A given problem is classified by developing "views" (technically, interpretations between theories) from the library problem-solving theory and the given problem domain theory. The views can be constructed incrementally by starting at the root of the hierarchy and developing views one level at a time. We have discovered four basic techniques for constructing views. We have shown that it subsumes all steps in our previously implemented tactics for applying problem-solving knowledge.

The key advantage of this approach is the relative ease with which a user can add new problem-solving knowledge to the system. There is no need to add special-purpose tactics for applying the knowledge – the tactics arise from the general tools for constructing views.

We have partially implemented this classification approach and have tested it on some simple problems. Jim McDonald has partially developed a Theory Morphism Construction Interface. It shows source and target theory presentations and the current (possibly partial) morphism between them. Users have several tools to support the completion of a morphism, including typing in translations for various source theory symbols and using "unskolemization" (one of the four basic methods mentioned above). We demonstrated the use of this system to develop a morphism from divide-and-conquer theory into a simple problem theory.

## 12. Applications to Scheduling Problems

We have begun to explore scheduling as an application area to build up in KIDS. I developed a formal theory of transportation scheduling defining the concepts of this problem (and similar ones) and developed laws for reasoning about them. This was about 18 pages long.

Because it was required for scheduling, I extended global search theory to handle constraint propagation. This was a difficult piece of original work – I have not seen any other model of search that captures constraint propagation in as general and precise a way. The resulting theory is simple, but gives rise to seemingly difficult inference problems. We are currently working to simplify these inferences. It also gives rise to specifications requiring an algorithm theory that is not currently in the system (fixpoint iteration).

We used KIDS to derive a global search algorithm for transportation scheduling that performs constraint propagation. To get around some of the hard problems related to synthesis of constraint propagation code, I simply supplied the result as if the needed tactic were present. We also optimized the code using KIDS.

The resulting scheduler was able to schedule the 500 movements in about 2 seconds. The schedule used minimal resources and satisfied all specified constraints. This improved dramatically on the scheduling time and resource utilization of a hand-crafted scheduler that Kestrel researchers produced last year. The major differences are that the older scheduler compiled constraints as demons and used a Simplex algorithm to check constraints. Thus constraint checking and propagation were performed by general-purpose tools. For the newer scheduler, we synthesized problem-specific constraint checking and propagation code that, as expected, ran much faster.

More recently the scheduler has found complete feasible solutions to transportation problems ranging up to 3500 movement requirements in under 3 minutes.

## References


[1] LOWRY, M. R. *Algorithm Synthesis Through Problem Reformulation.* PhD thesis, Computer Science Department, Stanford University, 1989.

[2] SMITH, D. R. Top-down synthesis of divide-and-conquer algorithms. *Artificial Intelligence 27*, 1 (September 1985), 43–96. (Reprinted in *Readings in Artificial Intelligence and Software Engineering*, C. Rich and R. Waters, Eds., Los Altos, CA, Morgan Kaufmann, 1986.).

[3] SMITH, D. R. Applications of a strategy for designing divide-and-conquer algorithms. *Science of Computer Programming 8*, 3 (June 1987), 213–229.

[4] SMITH, D. R. Structure and design of global search algorithms. Tech. Rep. KES.U.87.12, Kestrel Institute, November 1987. to appear in *Acta Informatica.*

[5] SMITH, D. R. KIDS: A knowledge-based software development system. In *Automating Software Design*, M. Lowry and R. McCartney, Eds. MIT Press, Menlo Park, 1991, pp. 483–514.

[6] SMITH, D. R. Structure and design of problem reduction generators. In *Constructing Programs from Specifications*, B. Möller, Ed. North-Holland, Amsterdam, 1991, pp. 91–124.

[7] SMITH, D. R. Automated derivation of parallel sorting algorithms. In *Parallel Algorithm Derivation and Program Transformation*, R. Paige, J. Reif, and R. Wachter, Eds. Kluwer Academic Publishers, New York, 1992.

[8] SMITH, D. R. Constructing specification morphisms. Tech. Rep. KES.U.92.1, Kestrel Institute, January 1992. Submitted for publication.

[9] SMITH, D. R. Track assignment in an air traffic control system: A rational reconstruction of system design. In *Proceedings of the Seventh Knowledge-Based Software Engineering Conference* (McLean, VA, September 1992), pp. 60–68.

[10] SMITH, D. R. KIDS – a semi-automatic program development system. *IEEE Transactions on Software Engineering Special Issue on Formal Methods in Software Engineering 16*, 9 (September 1990), 1024–1043.

[11] SMITH, D. R., AND LOWRY, M. R. Algorithm theories and design tactics. *Science of Computer Programming 14*, 2-3 (October 1990), 305–321.

P.I. Name: Douglas R. Smith
Insitution: Kestrel Institute
Telephone: (415) 493-6871
E-mail: smith@kestrel.edu
Contract Title: Theory of Algorithm Structure and Design
Contract Number: N00014-90-J-1733
Reporting Period: 1 Oct 89 - 30 Sep 92

## 3. Lists of Publications, Presentations, and Reports

### 3.1. Publications

Lowry, M.R., Category Theory and Homomorphic Abstraction, Proceedings of Workshop on Change of Representation and Problem Reformulation, Menlo Park, CA, March 1990 (also appeared in Proceedings of Workshop on Algebraic Approaches to Problem Solving and Perception Tarrytown, New York, June 1990).

Lowry, M.R., Abstracting Domains With Hidden State, Proceedings of AAAI-90 Workshop on Automatic Generation of Approximations and Abstraction, Boston, MA, August 1990.

Lowry, M.R., Structure and Design of Local Search Algorithms, in *Automating Software Design*, Eds. M. Lowry and R. McCartney AAAI Press, Menlo Park, CA, 1991.

Lowry, M.R., Software Engineering in the 21st Century, in *Automating Software Design*, Eds. M. Lowry and R. McCartney AAAI Press, Menlo Park, CA, 1991.

Lowry, M. and McCartney, R., Editors, *Automating Software Design*, AAAI Press, Menlo Park, CA, 1991.

Smith, D.R., Automating the Development of Software, Proceedings of the NATO Symposium on Military Information Systems Engineering, Malvern, England, May 1990.

Smith, D.R., KIDS: A Semi-Automated Program Development System, IEEE Transactions on Software Engineering 16(9), Special Issue on Formal Methods, September 1990, 1024-1043.

Smith, D.R., Automating the Development of Software, Proceedings of the Fifth Annual Knowledge-Based Software Assistant Conference, Liverpool, New York, September, 1990, 13-27.

Smith, D.R. and Lowry, M.R., Algorithm Theories and Design Tactics, Science of Computer Programming 14(2-3), Special Issue on the Mathematics of Program Construction, October 1990, 305-321.

Smith, D.R., Theory-based Support for Software Development, in *Proceedings of the Workshop on Automating Software Design: Interactive Design*, AAAI-91, Anaheim, California, July 1991.

Smith, D.R., KIDS: A Semi-Automated Program Development System, abstract in *Proceedings of SIGSOFT 90*, Irvine, California, December, 1990.

Smith, D.R., KIDS: A Knowledge-Based Software Development System, in *Automating Software Design*, Eds. M. Lowry and R. McCartney, MIT Press, Menlo Park, 1991.

Smith, D.R., Structure and Design of Problem Reduction Generators, in *Constructing Programs from Specifications*, Ed. B. Möller, North-Holland, Amsterdam, 1991, 91-124.

Smith, D.R., Track Assignment in an Air Traffic Control System: A Rational Reconstruction of System Design, in *Proceedings of the Seventh Knowledge-Based Software Engineering Conference*, IEEE Computer Society Press, September, 1992, 60-68.

Smith, D.R., Automated Derivation of Parallel Sorting Algorithms, to appear in *Parallel Algorithm Derivation and Program Transformation*, Eds. R. Paige, J. Reif, and R. Wachter, Kluwer Academic Publishers, 1992.

Smith, D.R., Automating the Design of Algorithms, to appear in the IFIP State of the Art Seminar series, Springer-Verlag, Eds. B. Möller, H. Partsch, S. Schumann, 1992.

Smith, D.R., Constructing Specification Morphisms, submitted to Journal of Symbolic Computation, Special Issue on Automatic Programming, 1992.

Smith, D.R., Structure and Design of Global Search Algorithms, to appear in Acta Informatica.

### 3.2. Presentations

*Michael R. Lowry:*

Presented a talk entitled "Problem Reformulation through Abstraction, then Implementation", Price Waterhouse Technology Center, 16 Novemeber 1989.

Presented a talk entitled "Problem Reformulation through Abstraction, then Implementation", University of Ottawa Computer Science Colloquium, 21 November 1989.

Presented a talk entitled "Problem Reformulation through Abstraction, then Implementation", University of Toronto Computer Science Colloquium, 23 November 1989.

Presented a talk entitled "Problem Reformulation through Abstraction, then Implementation", AT&T Bell Laboratories, 29 November 1989.

Presented a talk entitled "Problem Reformulation through Abstraction, then Implementation", Rutgers University Computer Science Colloquium, 30 November 1989.

Presented a talk entitled "A Comparison of Approaches to Problem Reformulation", Rutgers University Machine Learning Seminar, 1 December 1990.

Presented an informal talk entitled "Kestrel's Approach to Software Refinement", British Computer Society Third Refinement Workshop, 11 January 1990.

Presented a talk entitled "Category Theory and Isomorphic Reformulation", Workshop on Change of Representation and Problem Reformulation, Menlo Park, 24 March 1990.

Presented a talk entitled "Problem Reformulation through Abstraction, then Implementation", Workshop on Change of Representation and Problem Reformulation Menlo Park, 24 March 1990.

Presented a talk entitled "Category Theory and Homomorphic Reformulation", Workshop on Algebraic Approaches to Problem Solving and Perception, Tarrytown, New York, 27 June 1990.

*Douglas R. Smith*

Presented a talk entitled "Automating the Design of Algorithms", Computer Science Seminar, UCLA, Los Angeles, 14 November 1989.

Presented a talk entitled "Automating the Design of Algorithms", Information Technology Division, Naval Research Laboratories, Washington, DC, 20 December 1989.

Presented a talk entitled "REFINE" and a Refine demo, Presented a talk entitled "Automated Algorithm Design" and KIDS demo, Naval Postgraduate School, Monterey, California, February 1990.

Presented a talk entitled "KIDS: Knowledge-Based Software Development", San Francisco State University, San Francisco, California, 19 April 1990.

Presented a talk entitled "Operations on Theories in KIDS", IFIP WG2.1 meeting, Burton Manor, Burton, England, May 1990.

Presented a talk entitled "Automating the Development of Software", Symposium on Military Information Systems Engineering, Royal Signals and Radar Establishment, Malvern, England, 8-10 May 1990.

Presented a talk and demo entitled "KIDS: Knowledge-Based Program Development", Stanford University, Palo Alto, CA, 24 May 1990.

Presented a talk entitled "Automating the Design of Algorithms", ONR Contractor's Workshop, Moscow, Idaho, 20-21 June 1990.

Presented a paper entitled "Automating the Development of Software", gave demonstrations of KIDS, and participated in a panel on KBSA/CASE tools, Fifth Annual Knowledge-Based Software Assistant Conference, Liverpool, New York, 25 September 1990.

Presented a talk entitled "Automating the Development of Software", and gave a demonstration of KIDS, Bell-Northern Research Ltd., Ottawa, Ontario, 1 October 1990.

Presented a talk entitled "Automating the Development of Software", and gave a demonstration of KIDS, IBM CANADA Laboratory, North York, Ontario, 2 October 1990.

Presented a talk entitled "Automating the Development of Software", and gave a demonstration of KIDS, University of Toronto, Ontario, 3 October 1990.

Presented a talk on "Solving Scheduling Problems Using AI & OR in a Transformational Framework", Panel on Scheduling, DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control, San Diego, California, 5-8 November 1990.

Invited demonstration of KIDS at SIGSOFT 90, Software Development Environments 4, Irvine, California, 3-5 December 1990.

Presented a talk entitled "Structure and Design of Problem Reduction Generators", IFIP WG2.1 meeting, Louvain-la-Neuve, Belgium, 7-11 January 1991.

Presented a talk entitled "Scheduling", DARPA Planning Initiative, Kick-off meeting, St Louis, Missouri, 5-7 February 1991.

Presented a talk entitled "Structure and Design of Problem Reduction Generators", IFIP TC2 Working Conference on Constructing Programs from Specifications, IFIP Working Group 2.1, Pacific Grove, California, May 1991.

Presented a talk entitled "Theory-Based Support for Software Development", Workshop on Automating Software Design: Interactive Design, Anahiem, California, 15 July 1991.

Presented a talk entitled "Automating the Design of Algorithms", Workshop on Parallel Algorithm Derivation and Program Transformation, New York, New York, September 1991.

Presented a talk entitled "Integrating AI and OR: Some Foundations", Joint Operations Research/Artificial Intelligence Workshop on Transportation Planning, 30 Sept - 2 Oct 91, Carnegie Mellon University, Pittsburgh.

Presented a KIDS demo and talk entitled "Constructing Theory Interpretations", Principles of Programming Seminar, Computer Science Dept., Carnegie Mellon University, 2 Oct 91.

Presented a talk entitled "Data Structure Design", Kestrel Institute, Palo Alto, California, 13 Dec 91.

Panelist, AI and Software Engineering Panel, 1991 IEEE Conference on Tools for Artificial Intelligence, San Jose, California, 13 November 1991.

Presented KIDS demos, DARPA Software PI meeting, Los Angeles, 28-30 April 1992.

Presented a talk entitled "Algorithm Design via Classification" and KIDS demo, SRI International, Menlo Park, California, 4 May 92.

Presented a talk entitled "Algorithm Design via Classification" and KIDS demos, ONR PI meeting on Types and Algorithms, New Orleans, 18-20 May 1992.

Presented a talk and KIDS demo, Workshop on Advanced Technology for JMASS, Wright-Patterson AFB, Ohio, 5-6 Aug 1992.

Presented a KIDS demo and gave a talk entitled "Track Assignment in an Air Traffic Control System: A Rational Reconstruction of System Design", Seventh Knowledge-Based Software Engineering Conference, McLean, VA, 21-23 Sep 1992.

Session chair and Panelist, Seventh Knowledge-Based Software Engineering Conference, McLean, VA, 21-23 Sep 1992.

Presented a talk and KIDS demo, AFOSR, Bolling AFB, 24 Sep 1992.

Presented a talk entitled "Automating the Development of Software" and KIDS demo, NSA, Fort Meade, MD, 25 Sep 1992.

### 3.3. Technical Reports

Smith, D.R., KIDS: A Semi-Automatic Program Development System, Technical Report KES.U.90.1, Kestrel Institute, Palo Alto, CA, April 1990, 48 pages.

Smith, D.R., Structure and Design of Problem Reduction Generators, Technical Report KES.U.91.4, Kestrel Institute, Palo Alto, CA, May 1991, 34 pages.

Smith, D.R., Track Assignment in an Air Traffic Control System: A Rational Reconstruction of System Design, Technical Report KES.U.91.7, Kestrel Institute, Palo Alto, CA, July 1991.

Smith, D.R., Automated Derivation of Parallel Sorting Algorithms, Technical Report KES.U.91.12, Kestrel Institute, Palo Alto, CA, November 1991, 16 pages.

Smith, D.R., Constructing Specification Morphisms, Technical Report KES.U.92.1, Kestrel Institute, Palo Alto, CA, January 1992, 38 pages.

P.I. Name: Douglas R. Smith
Insitution: Kestrel Institute
Telephone: (415) 493-6871
E-mail: smith@kestrel.edu
Contract Title: Theory of Algorithm Structure and Design
Contract Number: N00014-90-J-1733
Reporting Period: 1 Oct 89 - 30 Sep 92

**4. Description of Research Transitions and DoD Interactions**

The main "transition" of the ONR-sponsored work has been through our experimental development system, KIDS (described on the next page). We have recently prepared a KIDS Users Manual and have officially released Version 1.0 of KIDS. We have received many requests for the system from researchers in software automation. Copies of KIDS are now installed at Catholic University of Louvain, Belgium (Sintzoff, Ledru), Technishe Hochschule Darmstadt, Germany (Bibel, Kreitz), Naval Postgraduate School, Monterey (Luqi), Andersen Consulting, Chicago (DeBellis, Miralya), and Rutgers Univ. (Mostow), Information Sciences Institute, USC (Balzer, Feather). Pending requests for copies of KIDS have come from over 40 sites in North America and Europe.

We have been working with the US Transportation Command on using KIDS for transportation scheduling (see *Applications to Scheduling Problems* above). Negotations are underway to use KIDS in support of the JMASS (Joint Modeling and Simulation Systems) effort at Wright-Patterson AFB.

Dr. Smith made several presentations to DoD and Government personnel during the contract period:

Presented a talk entitled "Automating the Design of Algorithms", Information Technology Division, Naval Research Laboratories, Washington, DC, 20 December 1989.

Presented a talk entitled "REFINE" and a Refine demo; also presented a talk entitled "Automated Algorithm Design" and KIDS demo, Naval Postgraduate School, Monterey, California, February 1990.

Presented a talk entitled "Automating the Development of Software", Symposium on Military Information Systems Engineering, Royal Signals and Radar Establishment, Malvern, England, 8-10 May 1990.

Presented a paper entitled "Automating the Development of Software", gave demonstrations of KIDS, and participated in a panel on KBSA/CASE tools, Fifth Annual Knowledge-Based Software Assistant Conference, Rome Air Development Center, New York, 25 September 1990.

Presented a talk on "Solving Scheduling Problems Using AI & OR in a Transformational Framework", Panel on Scheduling, DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control, San Diego, California, 5-8 November 1990.

Presented a talk entitled "Scheduling", DARPA Planning Initiative, Kick-off meeting, St Louis, Missouri, 5-7 February 1991.

Presented KIDS demos, DARPA Software PI meeting, Los Angeles, 28-30 April 1992.

Presented a talk entitled "Algorithm Design via Classification" and KIDS demos, ONR PI meeting on Types and Algorithms, New Orleans, 18-20 May 1992.

Presented a talk and KIDS demo, Workshop on Advanced Technology for JMASS, Wright-Patterson AFB, Ohio, 5-6 Aug 1992.

Presented a talk and KIDS demo, AFOSR, Bolling AFB, 24 Sep 1992.

Presented a talk entitled "Automating the Development of Software" and KIDS demo, NSA, Fort Meade, MD, 25 Sep 1992.

P.I. Name: Douglas R. Smith
Insitution: Kestrel Institute
Telephone: (415) 493-6871
E-mail: smith@kestrel.edu
Contract Title: Theory of Algorithm Structure and Design
Contract Number: N00014-90-J-1733
Reporting Period: 1 Oct 89 - 30 Sep 92

## 5. Description of Software and Hardware Prototypes

The Kestrel Interactive Development System (KIDS) provides an open architecture for experimenting with the semi-automated development of formal specifications into correct and efficient programs. The system has components for performing algorithm design, deductive inference, program simplification, partial evaluation, finite differencing optimizations, data type refinement and other development operations. Although their application is interactive, all of the KIDS operations are automatic except the algorithm design tactics which require some interaction at present. Over sixty programs have been derived using the system and we believe that KIDS could be developed to the point that it becomes economical to use for routine programming. We are not currently working on commercializing this system – it is regarded purely as an experimental testbed.